Relatório do Trabalho Prático de POO: Sistema de Gestão de Obra de Construção Civil

Bruno Paiva
a31496@alunos.ipca.pt

IPCA - Licenciatura de Engenharia de Sistemas Informáticos

Dezembro 2025

Resumo

Este relatório descreve o desenvolvimento da **Fase 1** do trabalho prático da Unidade Curricular de Programação Orientada a Objetos (POO), focado no tema **Gestão de Obra de Construção Civil**. O objetivo principal é a aplicação do Paradigma Orientado a Objetos para modelar e gerir os recursos associados a uma obra, incluindo materiais, armazéns, mão de obra (própria e subcontratada), veículos e orçamentos. O documento detalha a estrutura de classes, a definição de interfaces para garantir contratos claros e polimorfismo, e a aplicação de exceções customizadas para um tratamento de erros robusto.

Conteúdo

Capítulo 1

Introdução

1.1 Motivação e Tema

O trabalho foi desenvolvido no contexto da UC de POO, com o propósito de aplicar conceitos de encapsulamento, herança, polimorfismo e abstração na resolução de problemas reais. O tema escolhido, **Gestão de Obra de Construção Civil**, envolve a organização e controlo de custos e recursos de obras. As entidades-chave modeladas incluem: a empresa (Business), a obra (ConstructionWork), materiais, armazéns, veículos, mão de obra e orçamentos.

1.2 Objetivos da Solução

Os principais objetivos atingidos nesta Fase 1 foram:

- Identificação e modelação das entidades do domínio do problema.
- Definição de interfaces (`ICostable`, `IDescribable`, etc.) para padronização de comportamentos.
- Implementação das classes principais com encapsulamento adequado (propriedades e modificadores de acesso).
- Utilização de Coleções Genéricas (`List<T>`) para gestão dinâmica de dados.
- Criação de exceções personalizadas para validação de regras de negócio.

Capítulo 2

Arquitetura e Estrutura da Solução

2.1 Estrutura do Projeto

A solução encontra-se organizada no namespace `Projeto_POO`, contendo todas as classes de definição de dados e lógica de negócio, bem como o ponto de entrada da aplicação (`Program.cs`).

2.2 Interfaces Fundamentais

As interfaces foram criadas para desacoplar as classes e permitir o tratamento genérico de objetos com características comuns:

- `ICostable`: Define que um objeto possui um custo monetário (`decimal Cost`). Implementado por `Material`, `Labor`, `Vehicle` e `ConstructionWork`.
- `IDescribable`: Garante que a entidade possui uma descrição ou nome (`string Description`). Implementado por `Material`, `Storage`, `Labor`, `Budget` e `ConstructionWork`.
- `IIdentifiable`: Define um identificador numérico único (`int Code`). Implementado por `Material`.
- `IStorable`: Define que um item possui uma quantidade de armazenamento (`int Quantity`). Implementado por `StorageItem`.
- `IDateable`: Garante que a entidade possui uma data associada (`DateTime Date`). Implementado por `Budget`.

2.3 Classes e Relações

A arquitetura do sistema baseia-se nas seguintes entidades:

2.3.1 Business (Empresa)

A classe de topo que representa a empresa de construção. Gere o nome da empresa e mantém uma lista de obras (`ConstructionWork`).

2.3.2 ConstructionWork (Obra)

A classe central e agregadora. Representa uma obra específica e implementa `IDescribable` e `ICostable`. Esta classe agrega listas de todos os recursos necessários:

- `List<Storage>` (Armazéns)
- `List<Labor>` (Mão de Obra)
- `List<Vehicle>` (Veículos disponíveis)
- `List<VehicleUsage>` (Registos de uso de veículos)
- `List<Budget>` (Orçamentos)

2.3.3 Recursos Materiais

- **Material**: Define as características intrínsecas de um produto (Código, Descrição, Custo Unitário). Implementa `IIdentifiable`, `IDescribable` e `ICostable`.
- **Storage**: Representa um armazém físico. Implementa `IDescribable` e contém uma lista de itens.
- **StorageItem**: Relaciona um `Material` com uma quantidade específica (`Quantity`). Implementa `IStorable`.

2.3.4 Recursos Humanos e Máquinas

- **Labor**: Representa a mão de obra, distinguindo entre própria e subcontratada através de um booleano (`Subcontracted`). Implementa `IDescribable` e `ICostable`.

- **Vehicle**: Define um equipamento ou viatura (Matrícula, Modelo, Custo/Hora). Implementa `ICostable`.
- **VehicleUsage**: Regista a utilização efetiva de um veículo na obra. Armazena o veículo associado e as horas de uso (arredondadas para cima).

2.3.5 Gestão Financeira

- **Budget**: Representa registos de orçamento com data, descrição e valor. Implementa `IDateable` e `IDescribable`.

Capítulo 3

Detalhes de Implementação

3.1 Encapsulamento

Todos os atributos das classes são privados (`private`), sendo o acesso feito exclusivamente através de propriedades públicas (`Properties`). Isso permite validações futuras nos `setters`, como visto na classe `VehicleUsage`, onde as horas são arredondadas automaticamente:

```
public decimal Hours
{
    get => _hours;
    set => _hours = Math.Ceiling(value);
}
```

3.2 Exceções Customizadas

Para garantir a integridade dos dados e um tratamento de erros específico, foram criadas classes de exceção no ficheiro `custom_exceptions.cs`:

- `InvalidQuantityException`: Para erros em quantidades (e.g., negativas).
- `MaterialNotFoundException`: Quando se tenta aceder a um material inexistente.
- `StorageNotFoundException`: Para erros na localização de armazéns.
- `VehicleNotFoundException` e `VehicleExistsException`: Para gestão da frota de veículos.

- `ConstructionWorkNotFoundException`: Para validar a seleção da obra ativa.

Capítulo 4

Conclusão

A Fase 1 do trabalho prático permitiu estabelecer o modelo do sistema de Gestão de Obras. A estrutura de classes criada respeita os princípios de Programação Orientada a Objetos, com uma clara separação de responsabilidades e uso intensivo de interfaces. A solução atual suporta a definição de todos os recursos necessários (materiais, mão de obra, veículos) e está preparada para a implementação da lógica de negócio mais complexa e persistência de dados na Fase 2.

4.1 Trabalho Futuro (Fase 2)

Para a próxima fase, os próximos passos incluem:

- **Implementação Final:** Refinar a lógica de negócio e as validações.
- **Aplicação Demonstradora:** Implementar os métodos no projeto `Projeto_POO` para exercer as funcionalidades pedidas.
- **Testes Unitários:** Implementar testes para garantir uma cobertura mínima de 50% do código.
- **Persistência de Dados:** Adicionar a funcionalidade de guardar e carregar a obra em ficheiros binários.

4.2 Link do Repositório GitHub

Poderá aceder ao código-fonte completo através do seguinte link:
`https://github.com/yesisr/TP_POO_Fase-1`